(19) Europäisches Patentamt
European Patent Office
Office européen des brevets

(11) Publication number: **0 435 476 A2**

(12) EUROPEAN PATENT APPLICATION

(21) Application number: **90313062.3**

(51) Int. Cl.5: **G06F 15/40**

(22) Date of filing: **30.11.90**

(30) Priority: **23.12.89 GB 8929158**

(43) Date of publication of application:
03.07.91 Bulletin 91/27

(84) Designated Contracting States:
DE FR GB IT NL SE

(71) Applicant: **INTERNATIONAL COMPUTERS LIMITED**
ICL House
Putney, London, SW15 1SW(GB)

(72) Inventor: **Brown, Anthony Peter Graham**
15 Bramblegate, Edgcumbe Park
Crowthorne, Berks RG11 6JA(GB)

(74) Representative: **Guyatt, Derek Charles Patents and Licensing International Computers Limited et al**
Six Hills House London Road
Stevenage, Herts, SG1 1YB(GB)

(54) Database system.

(57) A database system holds data in the form of a sequence of records, each record comprising one or more fields. The database can be interrogated by a search query, which specifies a particular logical combination of comparisons to be performed on specified fields of each record. Before the search commences, the search query is compiled to produce an optimised sequence of search code. Each comparison operation is assigned a cost, reflecting the cost in time to retrieve the required fields and to perform the comparisons, and is also assigned a probability, indicating the probability that the comparison will produce a true result. Each logical operation in the search query is then processed, to find the order of handling its arguments that gives the minimum expected cost, and the arguments are re-arranged into that order.

EP 0 435 476 A2

DATABASE SYSTEM

Background to the invention

This invention relates to database systems. More specifically, the invention is concerned with a method and apparatus for handling queries in such a system.

In general, a database system stores data in the form of records, each record consisting of a number of fields. One way of accessing data in such a system is to search the database for records that match a specified search query. The search query may consist of a particular logical combination of comparison operations to be performed on specified fields of each record, a record being retrieved only if the logical combination results in a "true" value for that record.

A problem with such an arrangement is that, if a query is very complex, each record will take a long time to process, and hence the execution of the search queries will be very slow.

The object of the present invention is to alleviate this problem.

Summary of the invention

According to the invention there is provided a data base system comprising means for storing a data base comprising a sequence of records, and means for interrogating the database with a search query comprising a logical combination of comparisons to be performed on each record, wherein the system comprises means for re-arranging the order of said comparisons to optimise the execution of said search query

By optimizing the search query, the processing time for each record is reduced, and hence the overall execution speed of the query is increased.

It should be noted that the optimization of the search query is performed once only, for example as part of a query compilation process before the search is actually executed. Hence, the optimization process does not incur any cost at run time, when the search is executed.

Brief description of the drawings

Figure 1 shows a data processing system embodying the invention.

Figure 2 is a flow chart of a compiler for optimizing a search query.

Figure 3 is a syntax tree representing a search query.

Figure 4 shows a compiler subroutine in greater detail.

Description of an embodiment of the invention

One data processing system in accordance with the invention will now be described by way of example with reference to the accompanying drawings.

Overview of the system

Referring to Figure 1, the data processing system comprises a host computer 10 and a search processor 11. The search processor is connected to a number of disc drive units 12 by way of a bus 13.

The disc drive units 12 hold a relational database comprising one or more files. Each file consists of a sequence of records, each comprising one or more fields. Each field represents either a numerical value or text, and may be of fixed length or variable length. In the case of a fixed length field, the length of the field is the same in all records, whereas in the case of a variable length field it may vary from record to record. The length of the variable length field is computable, for example from a length code at the start of the field.

In operation, the host computer 10 performs the main processing workload of the system, consisting of one or more applications programs 14. An application program may generate search queries, specifying searches to be performed on the data base. Each search query is processed by means of a query compiler 15, to generate a sequence of search instructions. which are passed to the search processor 11 for execution.

A search query consists of a logical combination of comparison operations to be performed on each record in a file. For example, a query may be in the form

((C1 AND C2) OR C3) AND (C4 OR C5)

where C1 – C5 represent individual comparison operations. Each comparison may involve comparing a specified field of the record with a predetermined search key, or may involve comparing two different fields within the record.

Query compiler

Referring now to Figure 2, this shows the query compiler 15 in greater detail.

The first step 20 of the query compiler is to analyse the query, to create a syntax tree, representing the logical structure of the query.

For example, the query

((C1 AND C2) OR C3) AND (C4 OR C5)

would produce the tree structure as shown in Figure 3.

The syntax tree consists of a set of nodes, interconnected by branches. Each node represents either a comparison (C1, C2 etc) or a logical operation (AND, OR). Each node is represented by the compiler as a data structure containing the following items:

OPERATOR TYPE (AND, OR, COMPARISON)
NUMBER OF BRANCHES (i.e. sub-nodes)
LEFTMOST BRANCH (pointer to sub-node)
OTHER BRANCHES (pointers to sub-nodes)

The next step 21 of the query compiler is to call a SET COSTS subroutine. The set costs subroutine assigns a cost and a probability to each node. The cost assigned to a node is an estimate of the expected cost (in processing time) of establishing whether that node will yield a true or false result. The probability assigned to a node is the probability of a true result.

As will be described in greater detail later, the cost of a logic node (AND, OR) depends on the costs and probabilities of its branches and on the order in which the branches are evaluated. The SET COSTS subroutine sorts the branches into the order that gives the least cost, rearranging the pointers in the syntax tree if necessary to ensure that the branches are evaluated in this order. (It is assumed that the branches of a logic node are always evaluated in order, starting from the leftmost branch). If a node has n branches, then there are n! ways of ordering the branches, and the SET COSTS subroutine determines the optimum permutation.

The next step 22 of the query compiler is to call a GEN CODE subroutine. This subroutine generates instructions for fetching the specified field or fields and performing the specified comparisons. It also inserts JUMP instructions so as to link these comparisons together in the specified logical structure.

For example, the instructions generated to implement the syntax tree structure shown in Figure 3 are of the following form.

```
                    C1
                    IF FALSE GO TO LABEL  :  a
                    C2
                    IF TRUE GO TO LABEL   :  b
          :a        C3
                    IF FALSE GO TO LABEL  :  d
          :b        C4
                    IF TRUE GO TO LABEL   :  c
                    C5
                    IF FALSE GO TO LABEL  :  d
          :c        RETURN "TRUE"
          :d        RETURN "FALSE"
```

Compiler routines for generating code for performing logical combinations of operations are known as such, and so it is not necessary to describe the GEN CODE subroutine in any further detail.

SET COSTS Subroutine

Referring now to Figure 4, this shows the SET COSTS Subroutine in greater detail. This is initially called in respect of the root node, and then calls itself recursively, so as to scan the whole of the syntax tree, so as to process each node in the tree.

(40). The first action is to test the node type and to branch according to whether it is a comparison node or a logic node.

(41). In the case of a comparison node, the next step is to assign a cost value and a probability value to the node.

The cost of a comparison node is equal to the cost of accessing the required field(s) for the comparison, plus the cost of executing the comparison. The cost of accessing a field depends on the structure of the record. In particular, it generally takes longer to access a field that is preceded by one or more variable length fields, since it is necessary to access all those preceding fields in sequence on order to locate the start of the required field. In general, the cost of accessing a field can be estimated by

$$a + nb$$

where a and b are constants and n is the number of preceding variable length fields. The cost of executing a comparison depends on the comparison type. Simple comparisons such as "equals" "greater than" and "less than" can be performed rapidly and hence are assigned low cost values. More complex string comparisons such as "contains" take much longer and hence are assigned relatively higher cost values.

The probability value assigned to the comparison node is an estimate of the probability that the comparison will yield a "true" result. This estimate is made on the basis of past experience of the outcomes of comparisons. In the absence of any information about previous outcomes, predetermined default values are used. Typical values are as follows:

Comparison type	Probability
Not equal	0.99
Greater than	0.5
Greater than or equal	0.5
Less than	0.5
Less than or equal	0.5
Equal	0.01
Starts with	0.01
Contains	0.1
NOT contains	0.9
Null	0.05
NOT null	0.95
Between	0.05
In set	0.25

(42). If the node is a logic node, the next step in the SET COSTS routine is to make a recursive call to itself, for each branch from the node. This will result in each of the sub-nodes being processed by SET COSTS, so as to assign costs and probabilities to those sub-nodes. It will be appreciated that if any of the sub-nodes is itself a logic node, this will lead to further nested calls to SET COSTS, and so on, until a comparison node is reached.

(43). When all the branches of the logic node have been processed, the branches of the node are re-arranged to minimise the cost of the logic node, and cost and probability values are assigned.

For a logic node, the cost depends on the costs and probabilities of its branches, and on the order in which those branches are evaluated. For simplicity, the case where there are only two branches will be considered here. The extension of three or more branches will be clear.

AND Nodes

Consider an AND node having two branches with costs c1, c2 and probabilities p1, p2. If branch 1 is evaluated first, then the expected cost is

$$c1 + p1.\ c2$$

whereas is branch 2 is evaluated first, the expected cost is

$$c2 + p2.\ c1$$

The SET COSTS routine determines which of these two possible orders of evaluation gives the least cost and, if necessary, swaps over the pointers in the syntax tree structure, so that the branch with the least cost becomes the leftmost branch and hence is evaluated first.

The cost assigned to the node is this least cost value.

Assuming that the results of the comparisons are statistically independent, the probability assigned to the AND node is p1. p2.

OR node

Consider an OR node having two branches with costs c1, c2 and probabilities p1, p2. If branch 1 is evaluated first, the expected cost is:

$$c1 + (1 - p1)\ c2.$$

if, on the other hand, branch 2 is evaluated first, the expected cost is:

$$c2 + (1 - p2)\ c1.$$

As before, SET COSTS determines the order of evaluation which gives the least cost, and swaps over the pointers if necessary, to ensure that the least cost branch is evaluated first.

Assuming the comparisons are statistically independent, the probability assigned to the OR node is:

p1 + p2 - p1. p2.

Claims

1. a data base system comprising means (12) for storing a data base comprising a sequence of records, and means (11) for interrogating the database with a search query comprising a logical combination of comparisons to be performed on each record, characterised by means (15) for re-arranging the order of said comparisons to optimise the execution of said search query

2. A system according to claim 1 wherein the means for re-arranging the order of said comparisons comprises:
   a) means (42) for assigning a cost to each comparison involved in the search query, indicating the cost in time required to retrieve the data for the comparison and to perform the comparison,
   b) means (43) for assigning a probability to each comparison, indicating the probability of the comparison producing a predetermined result, and
   c) means (43) for finding the order of performing said comparisons that gives the minimum expected cost, and re-arranging the comparisons into that order.

3. A system according to claim 1 wherein the system comprises a host computer (10) for generating said search queries, and wherein the means for interrogating the database comprises a dedicated search processor (11).

4. A system according to claim 3 wherein said means for re-arranging the order of said comparisons comprises compilation means (15) resident in the host computer.

5. A system according to claim 1 wherein said means for storing the database comprises a plurality of disc file units (12).

6. A method of compiling a search query comprising a logical combination of comparisons, to produce an optimised sequence of search instructions, the method comprising:
   a) assigning a cost to each comparison involved in the search query, indicating the cost in time required to retrieve the data for the comparison and to perform the comparison,
   b) assigning a probability to each comparison, indicating the probability of the comparison producing a predetermined result, and
   c) finding the order of performing said comparisons that gives the minimum expected cost, and re-arranging the comparisons into that order.

7. A method according to claim 6 wherein said logical combination of comparisons comprises a tree structure, and wherein the step of assigning a cost to each comparison is performed recursively on each node of said tree structure, starting from the root of the tree.

Fig. 1.

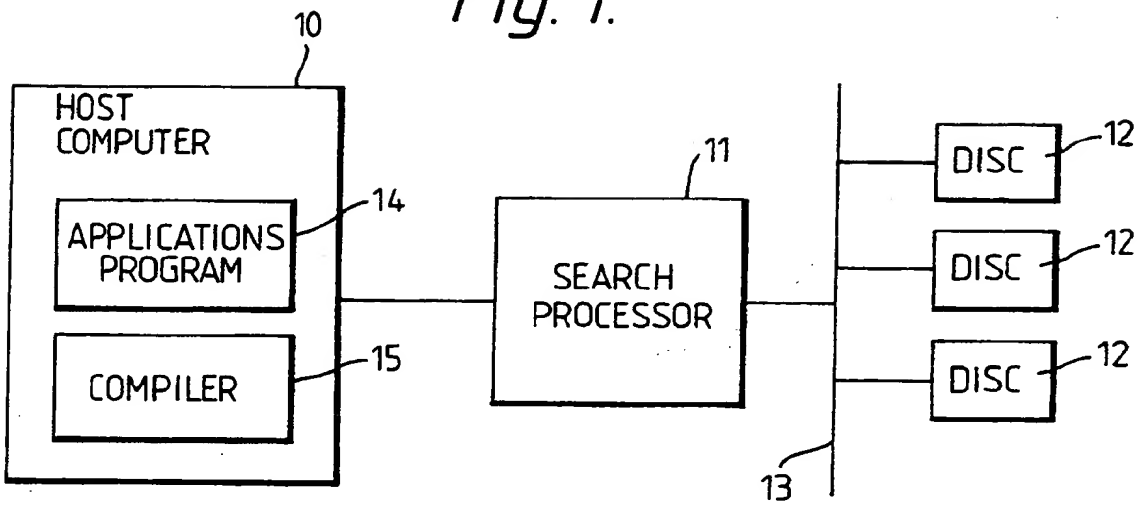

10
HOST COMPUTER
APPLICATIONS PROGRAM
14
COMPILER
15
11
SEARCH PROCESSOR
DISC
12
DISC
12
DISC
12
13

Fig. 2.

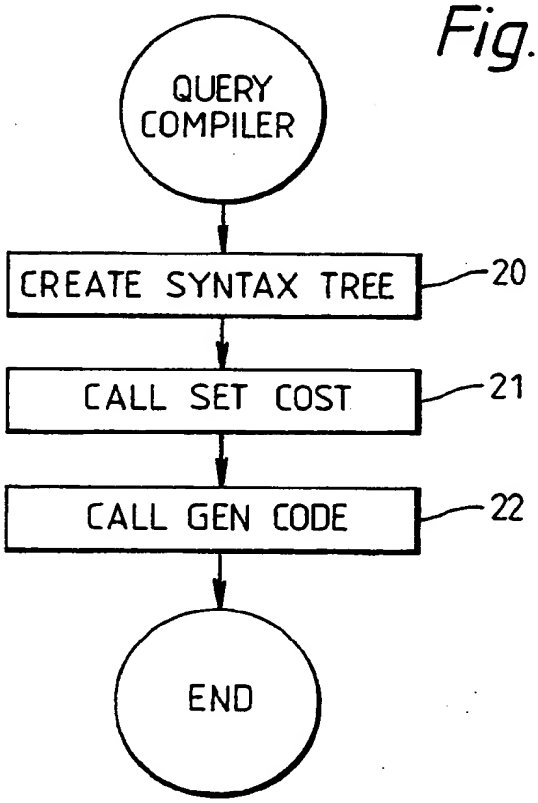

QUERY COMPILER
CREATE SYNTAX TREE
20
CALL SET COST
21
CALL GEN CODE
22
END

Fig. 3.

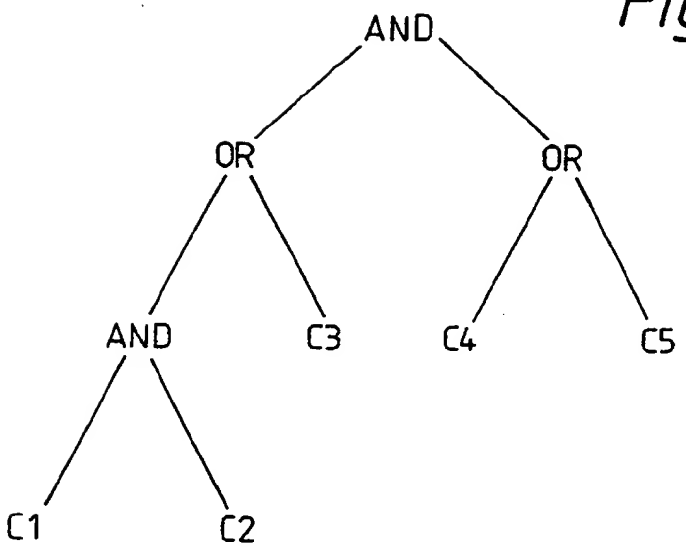
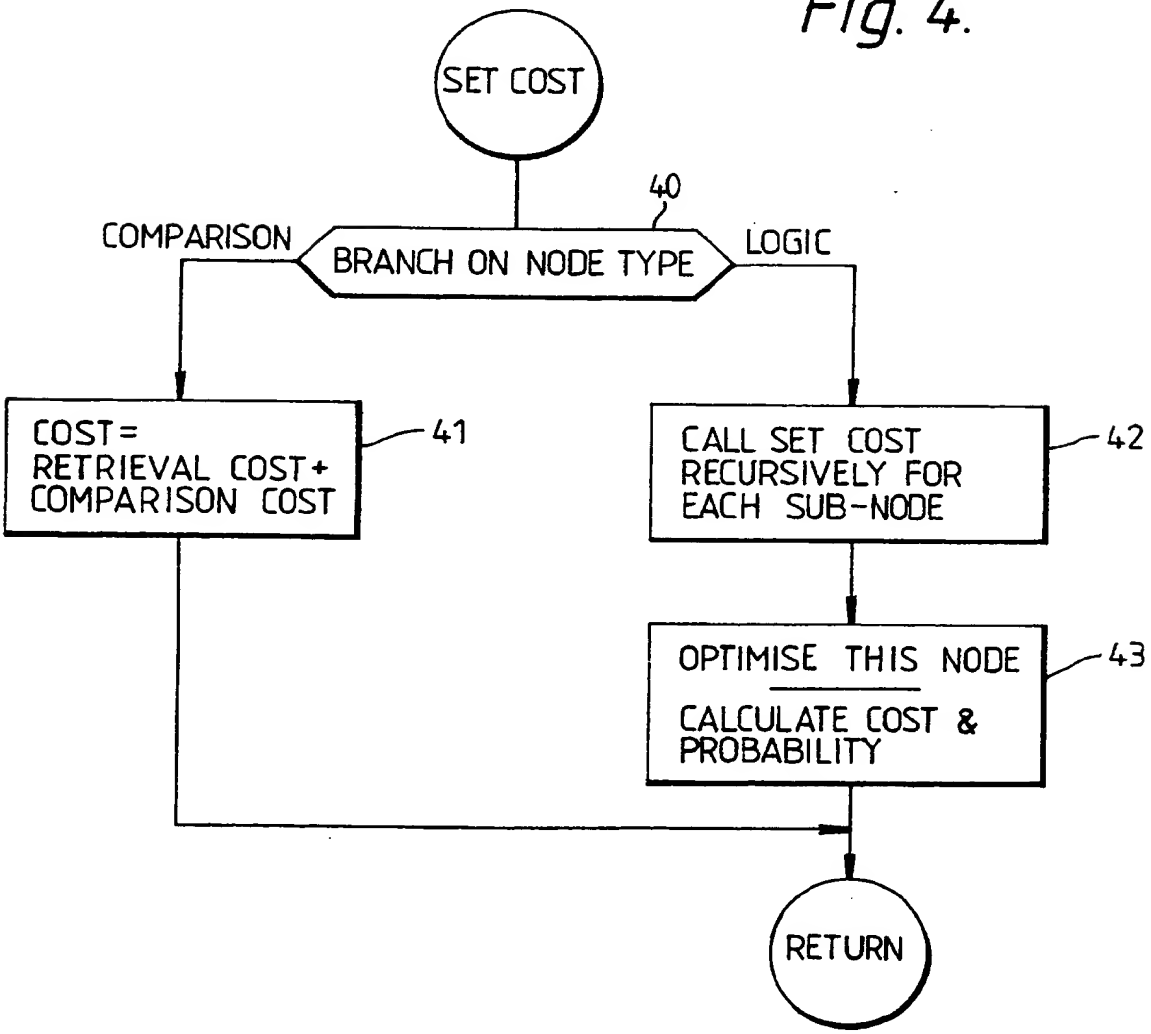


Fig. 4.



(19) Europäisches Patentamt
European Patent Office
Office eur pé n des brevets

(11) Publication number: **0 435 476 A3**

(12) EUROPEAN PATENT APPLICATION

(21) Application number: **90313062.3**

(51) Int. Cl.5: **G06F 15/403**, G06F 15/40

(22) Date of filing: **30.11.90**

(30) Priority: **23.12.89 GB 8929158**

(43) Date of publication of application:
03.07.91 Bulletin 91/27

(84) Designated Contracting States:
DE FR GB IT NL SE

(88) Date of deferred publication of the search report:
24.03.93 Bulletin 93/12

(71) Applicant: **INTERNATIONAL COMPUTERS LIMITED**
ICL House
Putney, London, SW15 1SW(GB)

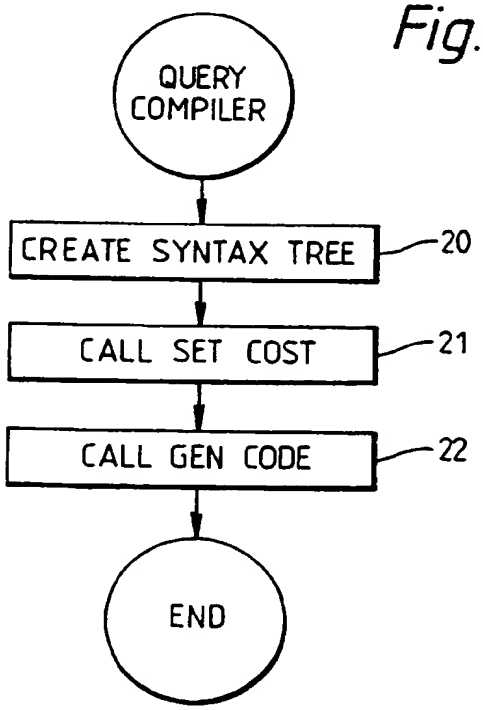
(72) Inventor: **Brown, Anthony Peter Graham**
15 Bramblegate, Edgcumbe Park
Crowthorne, Berks RG11 6JA(GB)

(74) Representative: **Guyatt, Derek Charles Patents and Licensing International Computers Limited et al**
Six Hills House London Road
Stevenage, Herts, SG1 1YB (GB)

(54) Database system.

(57) A database system holds data in the form of a sequence of records, each record comprising one or more fields. The database can be interrogated by a search query, which specifies a particular logical combination of comparisons to be performed on specified fields of each record. Before the search commences, the search query is compiled to produce an optimised sequence of search code. Each comparison operation is assigned a cost, reflecting the cost in time to retrieve the required fields and to perform the comparisons, and is also assigned a probability, indicating the probability that the comparison will produce a true result. Each logical operation in the search query is then processed, to find the order of handling its arguments that gives the minimum expected cost, and the arguments are rearranged into that order.

Fig. 2.



EP 0 435 476 A3

European Patent Office

EUROPEAN SEARCH REPORT

Application Number

EP 90 31 3062

DOCUMENTS CONSIDERED TO BE RELEVANT

Category	Citation of document with indication, where appropriate, of relevant passages	Relevant to claim	CLASSIFICATION OF THE APPLICATION (Int. Cl.5)
X	US-A-4 829 427 (N.L.GREEN) 9 May 1989	1	G06F15/403 G06F15/40
A	* abstract *	2-6	
A	NEC RESEARCH AND DEVELOPMENT no. 72, January 1984, TOKYO JP pages 48 - 55 S.HIYOSHI ET AL. 'Hierarchical optimization strategy for query evaluation' * page 48, column 1, line 21 - page 48, column 2, line 4 *	1-6	

TECHNICAL FIELDS SEARCHED (Int. Cl.5)

G06F

The present search report has been drawn up for all claims

Place of search	Date of completion of the search	Examiner
THE HAGUE	27 JANUARY 1993	KATERBAU R.E.

CATEGORY OF CITED DOCUMENTS

X : particularly relevant if taken alone
Y : particularly relevant if combined with another document of the same category
A : technological background
O : non-written disclosure
P : intermediate document

T : theory or principle underlying the invention
E : earlier patent document, but published on, or after the filing date
D : document cited in the application
L : document cited for other reasons

& : member of the same patent family, corresponding document

EPO FORM 1503 03.82 (P0401)